How to / Maniaplanet / Manialink / Build a Manialink

Author : Marcel @ Funtrackers

4.3 Event handling

Now that we know how to manipulate certain elements of the ManiaLink, of course we still need to know how to react on the user's input like mouse movement and clicks and key presses. For this, ManiaScript provides different kinds of events, which exactly represents these actions of the user.

- MouseClick – The user clicked with the left mouse button on an element.
- MouseOver – The user moved the mouse cursor into an element.
- MouseOut – The user moved the mouse out of an element.
- The user pressed a key.

There are some limitations on the events, which may hinder Events to be handled in a ManiaScript:

- Only events which are not handled by the element itself may be handled in a ManiaScript.
  This means, that e.g. an Entry will never create any KeyPress events, as it handles them itself by inserting the pressed character into the field. A linked Quad or Label will never create any mouse events.
- The mouse events (MouseClick, MouseOver and MouseOut) requires to have ScriptEvents set to 1 to be created for this element. If an element does not have ScriptEvents set to 1, it will not create any mouse events.
- Each event is triggered only once, the z-order decides about which element finally "gets" the event. For example a MouseClick will only be handled from the element which is above all others, i.e. from that one with the highest z-value in case of menu positioning or lowest z-value in case of classic positioning. All other elements behind this first one will not see the MouseClick at all.

Let's have a look on a typical event handling loop of a ManiaLink-ManiaScript:

Event handling of ManiaLink-ManiaScript:

```
main() {
// #1 Execute on initialization
while(True) {
// #2 Execute always foreach(Event in PendingEvents) {
switch(Event.Type) {
case CGameManialinkScriptEvent::Type::MouseClick: {
// #3 Execute on MouseClick event
}
case CGameManialinkScriptEvent::Type::MouseOver: {
// #4 Execute on MouseOver event } case CGameManialinkScriptEvent::Type::MouseOut:
{
// #5 Execute on MouseOut event } case CGameManialinkScriptEvent::Type::KeyPress:
{
// #6 Execute on KeyPress event
}
}
}
yield;
}
}
```

Basically we have six positions, where we can add some code to handle the different situations. If you want to execute some code once right after the ManiaLink finished loading, for example to hide some elements which should be showed later, do it outside the endless loop at position #1. In contrast to that, the code at position #2 will always be executed, this means right after it is done with this code, it will restart it again and again.

This is most suited for (time based) animations, as the more often they get executed the better and smoother they become. The foreach loop now is executed for each actual event triggered by any of the elements. You may use a switch to distinguish between their concrete types, as shown in the example with the positions #3 to #6. But this is up to you, finally, the only things you need is the infinite while loop, the foreach to handle all new events, and the yield at the end of the infinite loop.

4.4 Complex example

To make all this stuff a little bit clearer for you, I want to give you a full example, which demonstrates the use of a ManiaScript in a ManiaLink. The example is very basic, but still shows how to handle the Events without interfere an animation loop. If you have problems with understanding the example, you may simply copy&paste the XML code into a file, modify it a little bit and see in what these changes will result.

In detail, our example should display a continuous animation, which will be a simple Quad moving in a circle. Additionally, this Quad should react on the user's mouse: When receiving a MouseClick, MouseOver or MouseOut, the Quad should change its color to visualize these events. Let's have a look on the XML code itself:

XML code of the ManiaScript example:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<manialink version="1" background="0">
<timeout>0</timeout>
<!-- The container is used to change the positions of all Quads simultanously -->
<frame id="container">
<!-- The target Quad triggers the Mouse events we want to react on -->
<quad id="target" posn="0 0 5" sizen="20 20" ScriptEvents="1" />
<!-- As we cannot change the bgcolor of a Quad, we use multiple Quads and only display that one with the wanted color -->
<quad id="red" sizen="20 20" bgcolor="A00A" />
<quad id="green" sizen="20 20" bgcolor="080A" />
<quad id="blue" sizen="20 20" bgcolor="00AA" />
</frame>
<script><!--
#Include "MathLib" as MathLib

// Shows the Quad with the specified ID and hides all others (except the target)

Void ShowQuad(Text ControlID) {
declare CGameManialinkFrame Container <=> (Page.MainFrame.Controls["container"] as CGameManialinkFrame);
foreach(Quad in Container.Controls) {
if (Quad.Id == "target" || Quad.Id == ControlID) {
Quad.Show(); }
else { Quad.Hide();
}
}
}
// Initializes the animation
Void Initialize() {
// Save the time of initialization as we need it for the animation
declare Integer StartTime for Page = CurrentTime;
// Show the red Quad as default on initialization
ShowQuad("red"); }
// Calculates the current animation state and applys it to the container
Void Animate() {
// Make the previously saved StartTime available in our function
declare Integer StartTime for Page;
// Calculate the time difference in seconds
declare Real TimeDiff = (CurrentTime - StartTime) / 1000.;
// Update the current Position of the Container to finally animate it
declare CGameManialinkControl Container <=> Page.MainFrame.Controls["container"];
Container.PosnX = 30 * MathLib::Sin(TimeDiff);
Container.PosnY = 30 * MathLib::Cos(TimeDiff); }

main() {
Initialize();
while(True) {
Animate();
foreach(Event in PendingEvents) {
switch(Event.Type) {
case CGameManialinkScriptEvent::Type::MouseClick: {
ShowQuad("blue"); }
case CGameManialinkScriptEvent::Type::MouseOver: { ShowQuad("green"); }
case CGameManialinkScriptEvent::Type::MouseOut: { ShowQuad("red");
}
}
}
yield;
}
}
--></script>
</manialink>
```

The first problem we need to solve is that we cannot change the bgcolor of a Quad out of a ManiaScript. That's why we use several Quads with the same size and same position but different colors, and we will only show that one which has the currently wanted color. Those Quads have the IDs red, green and blue.

Because we do not want to observe these three Quads for the mouse events, a fourth Quad target has been added, having the same size and same position, but have been placed in front of all other Quads (so it will definitely catch all the mouse events) and without any color to make it invisible. This Quad is the only element with ScriptEvents="1", so whenever a mouse event has been triggered, we can be sure it is from this Quad – We don't have to check the ControlId at all in this example.

As all these Quads always should have the same position, we put all of them in a Frame called container: We will always move this container to animate the Quads, so we can be sure that they will always stay on top of each other.

So let's have a look on the ManiaScript itself, which is the important part of our example. The script has mainly three functions: ShowQuad(ControlID) will expect either of our visible Quads and will exactly show this one Quad and hide all others, making sure our target will always be shown to be able to catch our mouse events. So for example ShowQuad("blue"); will show our blue Quad instead whatever Quad have been shown before. Initialize() is called once when the ManiaLink has been finished loading to initialize any values we need, in our case saving the CurrentTime we'll need later and showing the red Quad. Animate() will calculate and update the current position of the Quads to fit our circle animation, based on the time which have been passed (TimeDiff, in seconds). This function will be called whenever we have nothing else to do to make the animation as smooth as possible.

The main() part of the ManiaScript now implements the concrete behavior "around" our functions. As first line, we see the call of Initialize();, so we know that we will have the red Quad shown as soon as we enter the infinite loop. The infinite loop now is divided in three parts:

1. Update the animation with the call of Animate().
2. Handle any mouse events which arrived since the last time. Depending of the actual type of the event to be handled, we show a different colored Quad, so that we can observe what events have reached the Quad. For example the Quad will become blue when you click on it, but will become red once you move the mouse outside of it – Following the Quad with your mouse will keep it blue.
3. yield gives back the control to ManiaPlanet, so the game will be able to re-fill the PendingEvents array (if some events have been triggered) and do other stuff... like finally displaying the changed position of our Quad. As soon as ManiaPlanet is done, the control is taken back to the ManiaScript (which will take only milliseconds so far) starting the next run of the infinite loop.

We can see that one event type is not handled at all: KeyPress. Take it as a little exercise: Expand the ManiaScript (and the ManiaLink itself) to additionally change the color of the Quad, let's say to yellow, if the user presses any key.

This site is friendly to stationary (PC/MAC) and mobile (Android/iOS) browsers
Trackmania, Shootmania, Questmania games & media © Nadeo & UbiSoft used under fair use policy
8,373,040 page views since January 2008. All tutorials, services, coding © Mania Creative 2008-2013